Discrete Sequence Filters

A kind of algebra for the representation of information in multipurpose environments

Damian Fascio

Abstract:

Discrete Sequence Filters (DSF) are a computational logical system based on information representations, which work with scarce data. Through their filtering operations, the DSF perform computational calculations that can be used in various fields of mathematics, science and technology, among others. Due to their simplicity, DSFs can considerably reduce the amount of software and hardware needed to perform calculations, which leads to a decrease in energy use and greater care for the environment.

Thanks to its versatility and the property of performing complex computational calculations based on elementary logical operations, DSFs could be used in various tasks. However, its primary purpose is to achieve the emergence of rudimentary artificial intelligence with the ability to learn within a scarce data environment.

DSF consists of 11 types of operations, based on 7 computational operators: **EMP**, **ONL**, **ONR**, **RAN**, **STR**, **MAJ**, **REC**; and 4 information transformations: **attach**, **extract**, **write**, **erase**.

Sequence structure:

A sequence is a series of discrete steps, and each step consists of the following:

input: Where the elements (with and without information) enter.

database: Where these elements are accumulated.

operator: The operator used to perform a specific filtering operation.

output: Where a copy of the element resulting from the filtering operation is obtained.

value: The numerical value of the resulting element.

remain: Where a copy of the remaining elements is obtained (excluding the element resulting from the filtering operation).

Elements and values:

1. **e**: Element without information. Numerical value: 0 (zero).

2. **ep**: Element with neutral information. Numerical value: 0 (zero).

3. **epX**: Element with specific information "X". Numerical value: 1

4. **epY**: Element with specific information "Y". Numerical value: -1

Origin of element nomenclature:

The letter "e" comes from the word "envelope", and the letter "p" from the word "paper".

We could make an analogy of the performance of the elements within a database as empty envelopes that enter an urn (represented as elements **e**), sometimes containing a blank paper inside (represented as elements **ep**), and other times containing a paper written inside, with a certain information "X" or "Y" (represented as elements **epX** or **epY** respectively).

Hierarchy of elements according to their strength:

By default, the natural strength of each element is as follows:

Elements without information (**e**): 0 (zero)

Elements with information (**ep**, **epX**, **epY**): 1

It is possible to arbitrarily add strength to elements with information. Example: **epX** = 2; **epY** = 1,7; etc.

As an example, if we have a database [**epX**, **epY**] where **epY** = 3, it means that this database is equivalent to [**epX**, **epY**, **epY**, **epY**].

It is not possible to add arbitrary strength to elements without information in any case.

About presence:

The "presence" of a species is determined by the quantities and strengths of its elements in the database. For this reason, the presence of a species is always equivalent to the total sum of those quantities and strengths. For example, in the database [**epX**, **epY**, **epY**] where **epX** = 2, both species have the same presence, so we could say that both species have 2 elements each (the species **epY** is the majority in number, but the species **epX** has a stronger element, and thus both species are on equal terms in terms of presence within the database (by nature the **RAN** operator will be activated in this case).

Operators:

Trivial Operators:

1. **EMP** (empty):

By nature, the **EMP** operator is automatically activated when a database is empty or contains only <elements without information>. This operator returns no result, or only a copy of an <element without information> present in the database. This operator can be used to check for the existence of an empty database, among other things.

2. **ONL** (only):

By nature, the **ONL** operator activates automatically when a database contains a single <element with information>. This operator returns as a result a copy of the single <element with information> present in such a database. This operator can be used to check for the existence of a single element with information within a database, among other things.

3. **ONR** (only repeated):

By nature, the **ONR** operator activates automatically when a database contains a single <element with information> repeatedly. This operator returns as a result a copy of the single <element with information> repeated in such a database. This operator can be used to check for the existence of repetition of a single <element with information> within a database, among other things.

Non-trivial Operators:

4. **RAN** (random):

By nature, the **RAN** operator activates automatically when a database contains <elements with information> of different species but of equal presence. Presence in the database is determined by two variables that act together: On the one hand, the number of elements, and on the other hand, the strength of those elements. When different species cannot be differentiated from each other in any way, the **RAN** operator is activated. This operator returns as a result a copy of any <element with information> present in such a database. This operator can be used to randomly choose an element with information, among other things.

5. **STR** (stronger):

By nature, the **STR** operator activates automatically when a database contains a species of <element with

information> of greater arbitrary strength. This operator returns as a result a copy of the strongest <element with information> in such a database. This operator can be used to check which is the strongest <element with information> in the database.

6. **MAJ** (majority):

By nature, the **MAJ** operator activates automatically when a database contains a species of <element with information> of largest number. This operator returns as a result a copy of the majority <element with information> in such a database. This operator can be used to check for the existence of a more abundant element within a database, among other things.

7. **REC** (recent):

The **REC** operator does not activate automatically but rather by arbitrary decision, when a database contains at least one <element with information>. This operator returns as a result a copy of the most recent <element with information> in such a database. This operator can be used to obtain the newest element within a database, among other things.

Operators' working modes:

By nature: Operators work automatically in the presence of a specific database (described above).

By decision: Operators are forced to work on any other type of database they can (see "Using operators by decision").

Post-operations transformations:

1. **attach**: Attach neutral or specific information. This post-operation transforms an element **e** into an element **ep**, **epX** or **epY**. This post-operations is the inverse of **extract**.

2. **extract**: Extract neutral or specific information. This post-operation transforms an element **ep**, **epX** or **epY** into an element **e**. This post-operations is the inverse of **attach**.

3. **write**: Write specific information. This post-operation transforms an **ep** element into an **epX** or **epY** element. This post-operations is the inverse of **erase**.

4. **erase**: Erase specific information. This post-operation transforms an **epX** or **epY** element into an **ep** element. This post-operations is the inverse of **write**.

Ideal database examples:

1. It does not contain any type of element. Example: []

2. Contains a single element with information. Example: [**epX**]

3. Contains only the repetition of a single element with information. Example: [**epX**, **epX**]

4. It contains the same amount of elements with information, of different species and equal strength. Example: [**epX**, **epY**] (in which **epX** = **epY**)

5. It contains the same amount of elements with information, of different species and different strengths. Example: [**epX**, **epY**] (in which **epX** > **epY**)

6. Contains an element with information that is more abundant than another. Example: [**epX**, **epX**, **epY**]

7. Contains two elements with information that are more abundant than another. Example: [**epX**, **epX**, **epY**, **epY**, **ep**]

Sequence types:

One-dimensional sequences: These are sequences that work autonomously.

Two-dimensional sequences: These are sequences that work in combination with other sequences.

Three-dimensional sequences: They are sequences that work at different levels of depth.

Number system used:

Balanced ternary.

One-dimensional sequences

Sequences with specific information elements (epX and epY)

The **epX** and **epY** elements have the same hierarchy by default and contain specific information. For example, the **epX** element might contain one instruction and the **epY** element might contain an opposite instruction. The numeric value of the **epX** element is always 1 and the numeric value of the **epY** element is always -1. If the database is empty, the numerical value of the result is always 0 (zero).

Sequence example #1

operators used: **EMP**, **ONL**, **ONR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

Sequence example #2

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**

step: 4

input: **epX**

database: [**epX**, **epY**, **epX**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

Sequence example #3

operators used: **EMP**, **ONL**, **REC**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **REC**

output: **epY**

value: -1

remain: **epX**

Sequence example #4

operators used: **EMP**, **ONL**, **STR**

Arbitrary strength: **epX** = 2

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **STR**

output: **epX**

value: 1

remain: **epY**

* In this case, the strongest element in the database is returned as the output element (**epX**). As we see, due to the differences in strengths between the elements, the database [**epX**, **epY**] is simply equivalent to a database [**epX**, **epX**, **epY**].

Sequences with neutral information elements (ep)

The **ep** element has the same hierarchy by default as the **epX** and **epY** elements, but its information is neutral. For example, if **epX** represents a specific instruction and **epY** represents another specific instruction, **ep** would represent the "do nothing" instruction. Its numerical value is always 0 (zero).

Sequence example #1

operators used: **EMP**, **ONL**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **ep**

database: [**epX**, **ep**]

operator: **RAN**

output: **ep**

value: 0

remain: **epX**

Sequence example #2

operators used: **EMP**, **ONL**, **STR**

Arbitrary strength: **ep** = 2

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **ep**

database: [**epX**, **ep**]

operator: **STR**

output: **ep**

value: 0

remain: **epX**

* In this case, the strongest element in the database is returned as the output element (**ep**), and again the database [**epX**, **ep**] is equivalent to [**epX**, **ep**, **ep**].

Sequences with empty information elements (e)

The element **e** has a lower hierarchy by default than the other elements **epX**, **epY** and **ep**. Unlike these, by nature element **e** does not contain any information and its numerical value is always 0 (zero).

Sequence example #1

operators used: **EMP**, **ONL**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **e**

database: [**epX**, **e**]

operator: **ONL**

output: **epX**

value: 1

remain: **e**

step: 4

input: **e**

database: [**epX**, **e**, **e**]

operator: **ONL**

output: **epX**

value: 1

remain: **e**, **e**

step: 5

input: **epY**

database: [**epX**, **e**, **e**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**, **e**, **e**

***** As we see, the element **e** is irrelevant in these cases, such that a database [**epX**, **e**] is simply equivalent to a database [**epX**]. Elements **e** are ignored by all operators except the **EMP** operator.

Sequence example #2

operators used: **EMP**, **ONL**, **REC**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **e**

database: [**epX**, **e**]

operator: **REC**

output: **epX**

value: 1

remain: **e**

* In this case the **REC** operator also ignores the **e** element, and returns the element with information most recent from the database as output (**epX**).

Recalculating the random operation with an element e

Despite not containing any information, element **e** allows us to repeat the same operation without modifying the database with new information. This is especially important if we need operators such as **RAN** to give us a different result than their previous execution.

Sequence example #1

Operators used: **EMP**, **ONL**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**

step: 4

input: **e**

database: [**epX**, **epY**, **e**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**, **e**

* In this case, the **RAN** operator is performed only between the elements with information in the database (**epX** and **epY**). The input of element **e** allows the **RAN** operator to be performed again, with the possibility that the output element is now different from the previous step.

Using operators by arbitrary decision

By nature, operators are automatically activated against a specific database. However, it is also possible to use them according to arbitrary decisions. All operators can be used by decision, except the **STR** and

MAJ operators. Any error in the result of an operator by decision automatically stops the sequence.

EMP operator:

The **EMP** operator can be used by arbitrary decision, as long as there is at least one element without information in the database, which already contains one or more elements with information. Otherwise, an error will be returned.

Sequence example #1

operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **e**

database: [**epX**, **e**]

operator: **EMP** (by decision)

output: **e**

value: 0

remain: **epX**

* In this case, the operator by nature of step 3 should be **ONL**. Using the **EMP** operator by decision in this case tells us that there is at least one empty element in the database. This is useful if we need to attach neutral or specific information to that element, among other things.

Sequence example #2

operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **e**

database: [**e**]

operator: **EMP**

output: **e**

value: 0

remain: -

step: 3

input: **ep**

database: [**e**, **ep**]

operator: **ONL**

output: **ep**

value: 0

remain: **e**

step: 4

input: **e**

database: [**e**, **ep**, **e**]

operator: **ONL**

output: **ep**

value: 0

remain: **e**, **e**

step: 5

input: **epX**

database: [**e**, **ep**, **e**, **epX**]

operator: **EMP** (by decision)

output: **e**

value: 0

remain: **ep**, **epX**

* In this case, the operator by nature of step 5 should be **RAN**.

Sequence example #3

operators used: **EMP**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **EMP** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 2 should be **ONL**.

ONL operator:

The **ONL** operator can be used by arbitrary decision, whenever there is a unique element with information of its species within the database, which already contains elements with information from

other species repeatedly. Otherwise, an error will be returned.

Sequence example #1

operators used: **EMP**, **ONL**, **ONR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY**

database: [**epX**, **epX**, **epY**]

operator: **ONL** (by decision)

output: **epY**

value: -1

remain: **epX**, **epX**

* In this case, the operator by nature of step 4 should be **MAJ**. The use of the **ONL** operator by decision in this case tells us that there is a unique element in its species within the database. This is useful if we need to find some kind of anomaly, or differentiate a main element from the rest (figure - background), among other things.

Sequence example #2

operators used: **EMP**, **ONL**, **ONR**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epX**

database: [**epX**, **epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 5

input: **ep**

database: [**epX**, **epX**, **epX**, **ep**]

operator: **MAJ**

output: **epX**

value: 1

remain: **ep**

step: 6

input: **ep**

database: [**epX**, **epX**, **epX**, **ep**, **ep**]

operator: **ONL** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 6 should be **MAJ**.

ONR operator:

The **ONR** operator can be used by arbitrary decision, whenever there are repeated elements with information of a single species within the database, which already contains at least one element with information of another species. Otherwise, an error will be returned.

Sequence example #1

operators used: **EMP**, **ONL**, **ONR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY**

database: [**epX**, **epX**, **epY**]

operator: **ONR** (by decision)

output: **epX**

value: 1

remain: **epY**

* In this case, the operator by nature of step 4 should be **MAJ**. It should be noted that in this database, both the **ONR** (by decision) and **MAJ** operators return the same result (**epX**). The use of the **ONR** operator by decision in this case indicates that there is a repetition of one of the elements within the database. This element could be considered the background of a main figure, among other things.

Sequence example #2

operators used: **EMP**, **ONL**, **ONR**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epX**

database: [**epX**, **epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 5

input: **epY**

database: [**epX**, **epX**, **epX**, **epY**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

step: 6

input: **epY**

database: [**epX**, **epX**, **epX**, **epY**, **epY**]

operator: **ONR** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 6 should be **MAJ**.

RAN operator:

The **RAN** operator can be used by arbitrary decision, as long as there is at least one element with information in the database. Otherwise, an error will be returned.

Sequence example #1

operators used: **EMP**, **ONL**, **ONR**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY**

database: [**epX**, **epX**, **epY**]

operator: **RAN** (by decision)

output: **epY**

value: -1

remain: **epX**, **epX**

* In this case, the operator by nature of step 4 should be **MAJ**. Using the **RAN** operator by decision in these cases is very useful if we work with probabilities. The **RAN** operator on this database has a 66.66% probability of returning **epX** as output and a 33.33% probability of returning **epY** as output.

Sequence example #2

operators used: **EMP**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **e**

database: [**e**]

operator: **RAN** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 2 should be **EMP**.

STR operator:

The **STR** operator cannot be used by arbitrary decision, returning an error in all cases.

Sequence example #1

operators used: **EMP**, **ONL**, **STR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **STR** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 3 should be **RAN**. Using the **STR** operator by decision in this case could help us check that all elements have the same strength within the database, among other things.

Sequence example #2

operators used: **EMP**, **ONL**, **STR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **e**

database: [**epX**, **e**]

operator: **STR** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 3 should be **ONL**. As we can see, an error occurs since the **STR** operator only works with arbitrarily assigned strengths, ignoring the default strengths of each element.

MAJ operator:

The **MAJ** operator cannot be used by arbitrary decision, returning an error in all cases.

Sequence example #1

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**

step: 4

input: **ep**

database: [**epX**, **epY**, **ep**]

operator: **MAJ** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 4 should be **RAN**. Using the **MAJ** operator by decision in this case could help us check that only unique elements of their species exist within the database, among other things.

Sequence example #2

operators used: **EMP**, **ONL**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **e**

database: [**epX**, **e**]

operator: **ONL**

output: **epX**

value: 1

remain: **e**

step: 4

input: **e**

database: [**epX**, **e**, **e**]

operator: **MAJ** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 4 should be **ONL**.

REC operator:

The **REC** operator is by default a decision operator, meaning it does not activate automatically for a specific database like the other operators. Contrary to what might be believed, the **REC** operator does not need the database to contain at least 2 elements with information, as it can also operate with a single element with information. In any other case, an error will be returned.

Sequence example #1

operators used: **EMP**, **REC**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **REC** (by decision)

output: **epX**

value: 1

remain: -

* In this case, the operator by nature of step 2 should be **ONL**. It should be noted that in this database, both the **REC** (by decision) and **ONL** operators return the same result (**epX**). Using the **REC** operator by decision in this case could serve us (like the **ONL** operator) to check the existence of an element with information when checking empty databases, among other things.

Sequence example #2

operators used: **EMP**, **REC**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **e**

database: [**e**]

operator: **REC** (by decision)

output: < error >

value: -

remain: -

* In this case, the operator by nature of step 2 should be **EMP**. We must always remember that the **REC**

operator works only with elements with information.

Post-operations transformations

Erase information from an element

The specific information of the **epX** and **epY** elements can be erased. In this case, these elements will be transformed into **ep** elements.

Sequence example #1

Operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2 (erase step)

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

--- Post-operation function ---

erase specific information from: **epX**

step: 3

input: **e**

database: [**ep**, **e**]

operator: **ONL**

output: **ep**

value: 0

remain: **e**

Extract information from an element

The specific information of an **epX**, **epY** element or the neutral information of an **ep** element can be extracted. In this case, these elements will be transformed into **e** elements.

Sequence example #1

Operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2 (extract step)

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

--- Post-operation function ---

extract specific information from: **epX**

step: 3

input: **e**

database: [**e**, **e**]

operator: **EMP**

output: **e**

value: 0

remain: -

Sequence example #2

Operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2 (extract step)

input: **ep**

database: [**ep**]

operator: **ONL**

output: **ep**

value: 0

remain: -

--- Post-operation function ---

extract neutral information from: **ep**

step: 3

input: **e**

database: [**e**, **e**]

operator: **EMP**

output: **e**

value: 0

remain: -

Write information to an element

Specific information from an **epX** or **epY** element can be written to an **ep** element, as long as these elements are in the same database. In this case an **ep** element will be transformed into an **epX** or **epY** element.

Sequence example #1

operators used: **EMP**, **ONL**, **REC**, **ONR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2 (copy step)

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

--- Post-operation function ---

copy specific information from: **epX**

step: 3 (write step)

input: **ep**

database: [**epX**, **ep**]

operator: **REC**

output: **ep**

value: 0

remain: **epX**

--- Post-operation function ---

write specific information to: **ep**

step: 4

input: **e**

database: [**epX**, **epX**, **e**]

operator: **ONR**

output: **epX**

value: 1

remain: **e**

* In this case the specific information of the **epX** element is copied and then written to the **ep** element, transforming it into another **epX** element.

Attach information to an element

The specific information of an element **epX**, **epY**, or the neutral information of an element **ep** can be attached to an element **e**, as long as these elements are in the same database, which entails an exchange of identities between both elements, and consequently also an exchange of their places in the database.

Sequence example #1

operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2 (extract step)

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

--- Post-operation function ---

extract specific information from: **epX**

step: 3 (attach step)

input: **e**

database: [**epX**, **e**]

operator: **EMP** (by decision)

output: **e**

value: 0

remain: **epX**

--- Post-operation function ---

attach specific information to: **e**

step: 4

input: **e**

database: [**e, epX**, **e**]

operator: **ONL**

output: **epX**

value: 1

remain: **e**, **e**

* In this case the places occupied by the elements **e** and **epX** in the database of step 3 are exchanged in the database of step 4.

Sequence example #2

operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: **-**

value: 0

remain: -

step: 2 (extract step)

input: **ep**

database: [**ep**]

operator: **ONL**

output: **ep**

value: 0

remain: -

--- Post-operation function ---

extract neutral information from: **ep**

step: 3 (attach step)

input: **e**

database: [**ep**, **e**]

operator: **EMP** (by decision)

output: **e**

value: 0

remain: **ep**

--- Post-operation function ---

attach neutral information to: **e**

step: 4

input: **e**

database: [**e, ep**, **e**]

operator: **ONL**

output: **ep**

value: 0

remain: **e**, **e**

* In this case the places occupied by the elements **e** and **ep** in the database of step 3 are exchanged in the database of step 4.

Sequences with feedback

The output, the remain, or both, can feed back to the input of the same sequence. This is especially useful for generating a loop repeating a state or create an alternation between two states, similar to a flip-flop circuit.

Note: When the input contains also new elements, the feedback elements enter after them.

Sequence example #1

(output as input)

Operators used: **EMP**, **ONL**, **ONR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX** (previous output)

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epX** (previous output)

database: [**epX**, **epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

* In this case the output of each step is always the same **epX** element, generating a loop.

Sequence example #2

(remain as input)

Operators used: **EMP**, **ONL**, **REC**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **REC**

output: **epY**

value: -1

remain: **epX**

step: 4

input: **epX** (previous remain)

database: [**epX**, **epY**, **epX**]

operator: **REC**

output: **epX**

value: 1

remain: **epY**

step: 5

input: **epY** (previous remain)

database: [**epX**, **epY**, **epX**, **epY**]

operator: **REC**

output: **epY**

value: -1

remain: **epX**, **epX**

step: 6

input: **epX**, **epX** (previous remain)

database: [**epX**, **epY**, **epX**, **epY**, **epX**, **epX**]

operator: **REC**

output: **epX**

value: 1

remain: **epY**, **epY**

step: 7

input: **epY**, **epY** (previous remain)

database: [**epX**, **epY**, **epX**, **epY**, **epX**, **epX**, **epY**, **epY**]

operator: **REC**

output: **epY**

value: -1

remain: **epX**, **epX**, **epX**, **epX**

step: 8

input: **epX**, **epX**, **epX**, **epX** (previous remain)

database: [**epX**, **epY**, **epX**, **epY**, **epX**, **epX**, **epY**, **epY**, **epX**, **epX**, **epX**, **epX**]

operator: **REC**

output: **epX**

value: 1

remain: **epY**, **epY**, **epY**, **epY**

* In this case the outputs of each step alternate the **epX** and **epY** element, creating an oscillation, similar to a flip-flop circuit.

Assigning arbitrary strength to a specific species

So far, elements with arbitrary strength have been entered into the database individually, but it is important to know that this strength it is present in all elements of its species. For this reason, if it is

indicated that **epX** = 3, every **epX** element that enters the database will have that same strength. We must remember that it is only possible to assign arbitrary strength to the elements **epX**, **epY** or **ep**.

Sequence example #1

Operators used: **EMP**, **ONL**, **STR**, **RAN**, **MAJ**

Arbitrary strength: **epX** = 3

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **STR**

output: **epX**

value: 1

remain: **epY**

step: 4

input: **epY**

database: [**epX**, **epY**, **epY**]

operator: **STR**

output: **epX**

value: 1

remain: **epY**, **epY**

step: 5

input: **epY**

database: [**epX**, **epY**, **epY**, **epY**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**, **epY**, **epY**

step: 6

input: **epY**

database: [**epX**, **epY**, **epY**, **epY**, **epY**]

operator: **MAJ**

output: **epY**

value: -1

remain: **epX**

step: 7

input: **epY**

database: [**epX**, **epY**, **epY**, **epY**, **epY**, **epY**]

operator: **MAJ**

output: **epY**

value: -1

remain: **epX**

step: 8

input: **epX**

database: [**epX**, **epY**, **epY**, **epY**, **epY**, **epY**, **epX**]

operator: **STR**

output: **epX**

value: 1

remain: **epY**, **epY**, **epY**, **epY**, **epY**

* In this case the total strength of the **epX** elements is equal to 6, which makes this species the strongest in the database.

Two-dimensional sequences

The appropriate combination of sequences in various numbers and shapes allows more complex computational calculations to be carried out.

Sequence example #1

Sequence **A**

Operators used: **EMP, REC**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epY**, **epY**, **epX**

database: [**epY**, **epY**, **epX**]

operator: **REC**

output: **epX**

value: 1

remain: **epY**, **epY**

Sequence **B**

Operators used: **EMP, ONR**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epY**, **epY** (sequence A remain)

database: [**epY**, **epY**]

operator: **ONR**

output: **epY**

value: -1

remain: -

Sequence **C**

Operators used: **EMP, MAJ**

step: 1

input: **-**

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **ep**, **epX**, **ep**, **e**

database: [**ep**, **epX**, **ep**, **e**]

operator: **MAJ**

output: **ep**

value: 0

remain: **epX**, **e**

Sequence **D**

Operators used: **EMP, ONR**

step: 1

input: **-**

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX** (sequence A output), **epX**, **e** (sequence C remain)

database: [**epX**, **epX**, **e**]

operator: **ONR**

output: **epX**

value: 1

remain: **e**

* In this case, sequence D has as raw material elements derived from the output and the remain of other

sequences.

Combining sequences to obtain other types of results

The combination of sequences also allows us to obtain other results, such as the least numerous element, or the element that is in the middle in terms of strength.

Sequence example #1

Sequence **A**

operators used: **EMP**, **ONL**, **STR**

Arbitrary strength: **epX** = 2

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **STR**

output: **epX**

value: 1

remain: **epY**

Sequence **B**

operators used: **EMP**, **ONL**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epY** (sequence A remain)

database: [**epY**]

operator: **ONL**

output: **epY**

value: -1

remain: -

* In this case, the least strong element of the last database of sequence A is obtained as output of sequence B (**epY**).

Sequence example #2

Sequence **A**

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**

step: 4

input: **epX**

database: [**epX**, **epY**, **epX**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

step: 5

input: **epY**

database: [**epX**, **epY**, **epX**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**, **epX**

step: 6

input: **epX**

database: [**epX**, **epY**, **epX**, **epY**, **epX**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**, **epY**

step: 7

input: **ep**

database: [**epX**, **epY**, **epX**, **epY**, **epX**, **ep**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**, **epY**, **ep**

Sequence **B**

operators used: **EMP**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epY**, **epY**, **ep** (sequence A remain)

database: [**epY**, **epY**, **ep**]

operator: **MAJ**

output: **epY**

value: -1

remain: **ep**

* In this case, the element that is between the most numerous and the least numerous in the last database of sequence A is obtained as output of sequence B (**epY**).

Sequence example #3

Sequence **A**

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **ep**

database: [**epX**, **ep**]

operator: **RAN**

output: **epX**

value: 1

remain: **ep**

step: 4

input: **epY**

database: [**epX**, **ep**, **epY**]

operator: **RAN**

output: **epX**

value: 1

remain: **ep**, **epY**

step: 5

input: **epY**

database: [**epX**, **ep**, **epY**, **epY**]

operator: **MAJ**

output: **epY**

value: -1

remain: **epX**, **ep**

Sequence **B**

operators used: **EMP**, **REC**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**, **ep** (sequence A remain)

database: [**epX**, **ep**]

operator: **REC**

output: **ep**

value: 0

remain: **epX**

* In this case, the penultimate element type of the last database of sequence A is obtained as output of sequence B (**ep**). We must note that the last element type contains two instances (**epY**, **epY**).

Specific range in random operation

It is possible that the execution of the **RAN** operator is carried out only on a specific range of the

database.

Majority elements random

Sequence example #1

Operators used: **EMP**, **ONL**, **ONR**, **MAJ**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY**

database: [**epX**, **epX**, **epY**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

step: 5

input: **epY**

database: [**epX**, **epX**, **epY**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**, **epX**

step: 6

input: **ep**

database: [**epX**, **epX**, **epY**, **epY**, **ep**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**, **epY**, **ep**

* In this case the **RAN** operator is not applied to the entire database but only to the majority elements in it (**epX** and **epY**). Something interesting happens here, and that is that in this case there is something similar to the superposition of two operators, since to obtain the final result the majority elements are first taken (where there would be an implicit **MAJ** operator) and after that a real **RAN** operator returns a random element as a result, which arises from that same majority group (**epX**). We will talk about implicit operators later.

Minority elements random

Sequence example #2

Sequence **A**

Operators used: **EMP**, **ONL**, **ONR**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY**

database: [**epX**, **epX**, **epY**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

step: 5

input: **ep**

database: [**epX**, **epX**, **epY**, **ep**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**, **ep**

Sequence **B**

Operators used: **EMP**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epY**, **ep** (sequence A remain)

database: [**epY**, **ep**]

operator: **RAN**

output: **epY**

value: -1

remain: **ep**

* In this case the **RAN** operator is not applied to the entire last database of sequence A but only to the minority elements in it (**epY** and **ep**).

Other examples containing two majority elements

Sequence example #1

Operators used: **EMP**, **ONL**, **RAN**, **MAJ**, **STR**

Arbitrary strength: **epY** = 2

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **ep**

database: [**ep**]

operator: **ONL**

output: **ep**

value: 0

remain: -

step: 3

input: **epX**

database: [**ep**, **epX**]

operator: **RAN**

output: **ep**

value: 0

remain: **epX**

step: 4

input: **epX**

database: [**ep**, **epX**, **epX**]

operator: **MAJ**

output: **epX**

value: 1

remain: **ep**

step: 5

input: **epY**

database: [**ep**, **epX**, **epX**, **epY**]

operator: **RAN**

output: **epX**

value: 1

remain: **ep**, **epY**

* Note how in this case the **RAN** operator only considers the majority elements in presence (**epX** and **epY**). Remember that the presence of a species is always defined by the variables of quantity and strength of its elements.

step: 6

input: **epY**

database: [**ep**, **epX**, **epX**, **epY**, **epY**]

operator: **STR**

output: **epY**

value: -1

remain: **ep**, **epX**, **epX**

* In this case, the **STR** operator returns the element of the species with the greatest presence (**epY**). This element is also among the majority elements (together with **epX**), but it might not be (see example below).

Sequence example #2

Operators used: **EMP**, **ONL**, **ONR**, **RAN**, **STR**

Arbitrary strength: **epY** = 2 and **ep** = 4

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY**

database: [**epX**, **epX**, **epY**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**

step: 5

input: **epY**

database: [**epX**, **epX**, **epY**, **epY**]

operator: **STR**

output: **epY**

value: -1

remain: **epX**, **epX**

step: 6

input: **ep**

database: [**epX**, **epX**, **epY**, **epY**, **ep**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**, **epX**, **ep**

* In this case it does not matter that the majority element species are **epX** and **epY**, since the **RAN** operator is activated when the presence of the **epY** and **ep** species are equal (both equal 4).

Sequence example #3

Operators used: **EMP**, **ONL**, **ONR**, **RAN**, **STR**, **REC**

Arbitrary strength: **epX** = 2 and **epY** = 4

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX**

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY**

database: [**epX**, **epX**, **epY**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**

step: 5

input: **epY**

database: [**epX**, **epX**, **epY**, **epY**]

operator: **STR**

output: **epY**

value: -1

remain: **epX**, **epX**

step: 6

input: **ep**

database: [**epX**, **epX**, **epY**, **epY**, **ep**]

operator: **REC**

output: **ep**

value: 0

remain: **epX**, **epX**, **epY**, **epY**

* In this case it does not matter that the majority element species are **epX** and **epY**, and even that they are stronger than the **ep** species, since the **REC** operator ignores these factors and simply returns the most recent element with information in the database (**ep**).

Parallel sequences

Two or more sequences can work in parallel. In these cases the outputs and/or the remain of each step of one sequence are taken as inputs in each step of another sequence. The main characteristic of parallel sequences is that in this case all the sequences work simultaneously (not successively).

Sequence example #1

Sequence **A**

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**

step: 4

input: **epY**

database: [**epX**, **epY**, **epY**]

operator: **MAJ**

output: **epY**

value: -1

remain: **epX**

Sequence **B** (simultaneous with sequence A)

operators used: **EMP**, **ONL**, **ONR**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX** (output from step 2 in sequence A)

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epX** (output from step 3 in sequence A)

database: [**epX**, **epX**]

operator: **ONR**

output: **epX**

value: 1

remain: -

step: 4

input: **epY** (output from step 4 in sequence A)

database: [**epX**, **epX**, **epY**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

Sequence **C** (also simultaneous with sequence A)

operators used: **EMP**, **ONL**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: - (remain from step 2 in sequence A)

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 3

input: **epY** (remain from step 3 in sequence A)

database: [**epY**]

operator: **ONL**

output: **epY**

value: -1

remain: -

step: 4

input: **epX** (remain from step 4 in sequence A)

database: [**epY**, **epX**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**

* In this case, sequences A, B and C work simultaneously. In parallel, each step of sequence B takes as input the output of each step of sequence A. In turn, each step of sequence C takes as input the remainder of each step of sequence A.

Overlapping operators

Sequences with alternative remain

Assigning different arbitrary strengths to elements of the same species

Until now, the remainder of the operators always excluded the entire species obtained as output, but it is also possible to assign different strength to elements of the same species and thereby obtain an alternative remainder that contains elements of the same species as the output species. As always up to this point, the presence of a species is defined by the quantity and strength of its elements, which is then compared to the presence of another species, and that continues to happen in this new section. Furthermore, in some of these examples we can notice a kind of overlap of operators.

Sequence example #1

Operators used: **EMP**, **ONL**, **RAN**, **STR**, **MAJ**

Implicit operators: (**MAJ**), (**STR**)

Arbitrary strength: **epY$_2$** = 2

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY$_1$**

database: [**epX**, **epY$_1$**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY$_1$**

step: 4

input: **epY$_2$**

database: [**epX**, **epY$_1$**, **epY$_2$**]

operator: (**MAJ**) **STR**

output: **epY$_2$**

value: -1

remain: **epX**, **epY$_1$** <---- (alternative remain)

* In this case, even though there is a majority species in the database (**epY**), the **STR** operator is activated because within that species there is also a stronger element (**epY$_2$**). Here, too, there is a kind of overlap of operators, since the final result arises from the majority group of elements (i.e. there would be an implicit **MAJ** operator). After that, a explicit **STR** operator returns the strongest element as a result, which arises from that same majority group (**epY$_2$**).

step: 5

input: **epX**

database: [**epX**, **epY$_1$**, **epY$_2$**, **epX**]

operator: (**STR**) **STR**

output: **epY$_2$**

value: -1

remain: **epX**, **epY$_1$**, **epX** <---- (alternative remain)

* In this case there is a kind of overlap of an implicit **STR** operator that defines the strongest species (**epY**), and another real **STR** operator that defines the strongest element within that same species (**epY$_2$**).

step: 6

input: **epX**

database: [**epX**, **epY$_1$**, **epY$_2$**, **epX**, **epX**]

operator: **RAN**

output: **epY$_1$**

value: -1

remain: **epX**, **epY$_2$**, **epX**, **epX** <---- (alternative remain)

* Note that in this case the **RAN** operator returns as a result the weakest element of the **epY** species (**epY$_1$**). This species is also the least numerous species in the database.

step: 7

input: **epX**

database: [**epX**, **epY$_1$**, **epY$_2$**, **epX**, **epX**, **epX**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY$_1$**, **epY$_2$**

* Note something interesting in this example: The remain contains no elements of the **epX** species. This

happens because all elements of this species have the same strength, and this prevents the generation of an alternative remain in this case.

Sequence example #2

Operators used: **EMP**, **ONL**, **STR**, **RAN**

Implicit operators: (**STR**)

Arbitrary strength: $\mathbf{epX_1} = 3$ and $\mathbf{epY_2} = 2$

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: $\mathbf{epX_1}$

database: [$\mathbf{epX_1}$]

operator: **ONL**

output: $\mathbf{epX_1}$

value: 1

remain: -

step: 3

input: $\mathbf{epY_1}$

database: [**epX$_1$**, **epY$_1$**]

operator: **STR**

output: **epX$_1$**

value: 1

remain: **epY$_1$**

step: 4

input: **epY$_2$**

database: [**epX$_1$**, **epY$_1$**, **epY$_2$**]

operator: **RAN**

output: **epY$_1$**

value: -1

remain: **epX$_1$**, **epY$_2$** <---- (alternative remain)

* Note that in this step the result of the **RAN** operator randomly returns the element with the least strength of its species (**epY$_1$**).

step: 5

input: **epX$_2$**

database: [**epX$_1$**, **epY$_1$**, **epY$_2$**, **epX$_2$**]

operator: (**STR**) **STR**

output: **epX$_1$**

value: 1

remain: **epY$_1$**, **epY$_2$**, **epX$_2$** <---- (alternative remain)

* Note that in this step the result of the **STR** operator returns the strongest element of its species (**epX$_1$**). In this case there is also a kind of overlap of an implicit **STR** operator that defines the strongest species

(**epX**), and another real **STR** operator that defines the strongest element within that same species (**epX$_1$**).

step: 6

input: **epY$_3$**

database: [**epX$_1$**, **epY$_1$**, **epY$_2$**, **epX$_2$**, **epY$_3$**]

operator: **RAN**

output: **epX$_2$**

value: 1

remain: **epX$_1$**, **epY$_1$**, **epY$_2$**, **epY$_3$** <---- (alternative remain)

* In this case, the **RAN** operator returns as a result and in a random manner the least strong element of its species, which in turn is the least numerous species (**epX$_2$**).

Sequence example #3

Sequence **A**

Operators used: **EMP**, **ONL**, **STR**, **REC**

Arbitrary strength: **epX$_1$** = 2

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX$_1$**

database: [**epX$_1$**]

operator: **ONL**

output: **epX$_1$**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX$_1$**, **epY**]

operator: **STR**

output: **epX$_1$**

value: 1

remain: **epY**

step: 4

input: **epX$_2$**

database: [**epX$_1$**, **epY**, **epX$_2$**]

operator: **REC**

output: **epX$_2$**

value: 1

remain: **epX$_1$**, **epY** <---- (alternative remain)

* In this case the **epX** element with the lowest strength is chosen as output through a **REC** operator (**epX$_2$**). As we see, the **REC** operator ignores the strongest element (**epX$_1$**) and only returns the most recent element with information as a result (**epX$_2$**). Something important to know: The **REC** operator is

not returning the most recent element within the majority species (**epX**), but within the entire database. For that reason, in this case before the **REC** operator there is no implicit **MAJ** operator. Decision operators always ignore the database configuration, that is, they do not take into account the presence of each species within it (remember that the presence is determined by the quantities and strengths of the elements of each species).

Sequence **B**

Operators used: **EMP**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **ep** (new element), $\mathbf{epX_2}$ (sequence A output)

database: [**ep**, $\mathbf{epX_2}$]

operator: **RAN**

output: **ep**

value: 0

remain: $\mathbf{epX_2}$

* In this case, the output of sequence A that enters as input of sequence B is the element $\mathbf{epX_2}$ without the arbitrary assignment of higher strength, such that in this sequence B $\mathbf{epX_2}$ = **ep**.

Adding operators by arbitrary decision

Sequence example #1

Sequence **A**

Operators used: **EMP**, **ONL**, **STR**

Arbitrary strength: $\mathbf{epX_1} = 4$ and $\mathbf{epY_2} = 2$

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: $\mathbf{epX_1}$

database: [$\mathbf{epX_1}$]

operator: **ONL**

output: $\mathbf{epX_1}$

value: 1

remain: -

step: 3

input: $\mathbf{epY_1}$

database: [$\mathbf{epX_1}$, $\mathbf{epY_1}$]

operator: **STR**

output: **epX$_1$**

value: 1

remain: **epY$_1$**

step: 4

input: **epY$_2$**

database: [**epX$_1$**, **epY$_1$**, **epY$_2$**]

operator: **STR**

output: **epX$_1$**

value: 1

remain: **epY$_1$**, **epY$_2$**

Sequence **B**

Operators used: **EMP**, **RAN**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epY$_1$**, **epY$_2$** (sequence A remain)

database: [**epY**$_1$, **epY**$_2$]

operator: **RAN** (by decision)

output: **epY**$_1$

value: -1

remain: **epY**$_2$ <---- (alternative remain)

* In this case, a **RAN** operator is used by decision in a database that contains elements of different strength, but belonging to the same species (**epY**). The natural operator in this case should be **STR**.

Three-dimensional sequences

Sequences within sequences

Since an **epX** or **epY** element can represent any type of information, it is also possible that these elements are able to represent the same sequence, another sequence, or even a group of two-dimensional sequences. In these cases we would be talking about sequences that contain other sequences within themselves, which implies different levels of nesting, and which in some cases also refers to the concept of fractal, in which the whole is reflected in each of the parts.

Sequence example #1

Sequence **A**

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**

step: 4

input: **epX**

database: [**epX**, **epY**, **epX**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

Sequence **B**

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epY**

database: [**epY**]

operator: **ONL**

output: **epY**

value: -1

remain: -

step: 3

input: **ep**

database: [**epY**, **ep**]

operator: **RAN**

output: **ep**

value: 0

remain: **epY**

step: 4

input: **epX**

database: [**epY**, **ep**, **epX**]

operator: **RAN**

output: **epX**

value: 1

remain: **epY**, **ep**

step: 5

input: **e**

database: [**epY**, **ep**, **epX**, **e**]

operator: **RAN**

output: **epY**

value: -1

remain: **ep**, **epX**, **e**

step: 6

input: **ep**

database: [**epY**, **ep**, **epX**, **e**, **ep**]

operator: **MAJ**

output: **ep**

value: 0

remain: **epY**, **epX**, **e**

Sequence **C**

operators used: **EMP**, **ONL**, **RAN**, **MAJ**

step: 1

input: -

database: []

operator: **EMP**

output: -

value: 0

remain: -

step: 2

input: **epX**

database: [**epX**]

operator: **ONL**

output: **epX**

value: 1

remain: -

step: 3

input: **epY**

database: [**epX**, **epY**]

operator: **RAN**

output: **epY**

value: -1

remain: **epX**

step: 4

input: **epX**

database: [**epX**, **epY**, **epX**]

operator: **MAJ**

output: **epX**

value: 1

remain: **epY**

* In this case, the element **epX** of sequence C represents the entire sequence A, and the element **epY** of sequence C represents the entire sequence B. This means that each time an **epX** element is output in a step of sequence C, sequence A will be executed in its entirety at a different level. Likewise, each time an **epY** element is output in a step of sequence C, sequence B will also be executed in its entirety at a different level. Note also that the structure of sequence C is identical to the structure of sequence A.

Final note:

These are just a few examples of autonomous sequences and combinations of basic sequences to understand the correct functioning of DSFs. The possibilities of combining sequences in multidimensional terms offer a wide range of computational calculations that can be used in various areas of mathematics, science and technology, among other disciplines. The emulation of the operation of logic gates, perceptrons (neural networks), cellular automata, a theoretical framework for a programming language or the creation of a Turing-complete system are some of the interesting challenges that could be carried out through the correct study and research of DSF.

Damian Fascio.-

Córdoba, Argentina, October, 17, 2024.